Visibility-based Decentralized Swarm Decision Making Algorithms in 3D Urban Environments

Oren Gal and Yerach Doytsher
Mapping and Geo-information Engineering
Technion - Israel Institute of Technology
Haifa, Israel
e-mails: {orengal,doytsher}@alumni.technion.ac.il

Abstract— **In this paper, we present a unique and efficient visible trajectory planning for aerial swarm using decentralized algorithms in a 3D urban environment. By using SwarmLab environment, we compare two decentralized algorithms from the state of the art for the navigation of aerial swarms, Olfati-Saber's and Vasarhelyi's. The first step in our concept is to extract basic geometric shapes. We focus on three basic geometric shapes from point clouds in urban scenes that can be appear: planes, cylinders and spheres, extracting these geometric shapes using efficient Random Sample Consensus (RANSAC) algorithms with a high success rate of detection. The second step is a decentralized swarm algorithms for motion planning, demonstrated on drones in urban environment. Our planner includes dynamic and kinematic platform's limitation, generating visible trajectories based on our first step mentioned earlier. We demonstrate our visibility and trajectory planning method in simulations, showing trajectory planning in 3D urban environments for drone's swarm with decentralized algorithms with performance analysis such as order, safety, connectivity and union.**

Keywords-Swarm; Visibility; 3D; Urban environment; Decentlized algorithms.

I. INTRODUCTION AND RELATED WORK

In this paper, we study a fast and efficient visible trajectory planning drone swarms in a 3D urban environment, based on local point clouds data. Recently, urban scene modeling has become more and more precise, using Terrestrial/ground-based LiDAR on unmanned vehicles to generate point clouds data for modeling roads, signs, lamp posts, buildings, trees and cars. Visibility analysis in complex urban scenes is commonly treated as an approximated feature due to computational complexity.

Our trajectory planning method is based on a two-step visibility analysis in 3D urban environments using predicted visibility from point clouds data. The first step in our unique concept is to extract basic geometric shapes. We focus on three basic geometric shapes from point clouds in urban scenes: planes, cylinders and spheres, extracting these geometric shapes using efficient RANSAC algorithms with a high success rate of detection. The second step includes decentralized swarm algorithms for motion planning, demonstrated on drones in urban environment. Our planner includes dynamic and kinematic platform's limitation, generating visible trajectories based on our first step mentioned earlier. We demonstrate our visibility and trajectory planning method in simulations, showing trajectory planning in 3D urban environments for drone's swarm with decentralized algorithms with performance analysis such as order, safety, connectivity and union [1].

Visibility analysis based on this approximated scene prediction is done efficiently, based on our analytic solutions for visibility boundaries. With this capability, we present a local on-line planner generating visible trajectories, exploring the most visible and safe node in the next time step, using our predicted visibility analysis.

For the first time, we propose a solution for decentralized swarm algorithm which takes visibility into account, avoiding obstacles using Velocity Obstacle (VO) search and planning method.

II. VISIBILITY ANALYSIS FROM POINT CLOUDS DATA

As mentioned, visibility analysis in complex urban scenes is commonly treated as an approximated feature due to its computational complexity. Recently, urban scene modeling has become more and more exact, using Terrestrial/ground-based LiDAR generating dense point clouds data for modeling roads, signs, lamp posts, buildings, trees and cars. Automatic algorithms detecting basic shapes and their extraction have been studied extensively and are still a very active research field [2].

In this part, we present a unique concept for predicted and approximated visibility analysis in the next attainable vehicle's state at a one-time step ahead in time, based on local point clouds data which is a partial data set.

We focus on three basic geometric shapes in urban scenes: planes, cylinders and spheres, which are very common and can be used for most urban entities in modeling scenarios. Based on point clouds data generated from the current vehicle's position in state k-1, we extract these geometric shapes using efficient RANSAC algorithms [3] with high success rate detection tested in real point cloud data.

After extraction of these basic geometric shapes from local point clouds data, our unified concept, and our main

contribution, focus on the ability to predict and approximate urban scene modeling at the next view point V_k, i.e., at the attainable location of the vehicle in the next time step. Scene prediction is based on the geometric entities and the KF, which is commonly used in dynamic systems for tracking target systems [4],[5]. We formulate the geometric shapes as states vectors in a dynamic system and predict the scene structure the in the next time step, k.

Based on the predicted scene in the next time step, visibility analysis is carried out from the next view point model [6], which is, of course, an approximated one. As the vehicle reaches the next viewpoint V_k, point clouds data are measured and scene modeling and states vectors are updated, which is an essential for the global swarm visible trajectory planning based on state-of-the-art decentralized algorithms.

A. Shapes Extraction

1) Geometric Shapes:

The urban scene is a very complex one in the matter of modeling applications using LiDAR, and the generated point clouds are very dense. Despite these inherent complications, feature extraction can be made very efficient by using basic geometric shapes. We define three kinds of geometric shapes: planes, cylinders and spheres, with a minimal number of parameters for efficient time computation.

Plane: center point (x,y,z) and unit direction vector from center point.

Cylinder: center point (x,y,z), radius and unit direction vector of the cylinder axis. Cylinder height dimension will be considered later on as part of the simulation.

Sphere: center point (x,y,z), radius and unit direction vector from center point.

2) RANSAC:

The RANSAC [7] is a well-known paradigm, extracting shapes from point clouds using a minimal set of a shape's primitives generated by random drawing in a point cloud set. Minimal set is defined as the smallest number of points required to uniquely define a given type of geometric primitive.

For each of the geometric shapes, points are tested to approximate the primitive of the shape (also known as "score of the shape"). At the end of this iterative process, extracted shapes are generated from the current point clouds data.

Based on the RANSAC concept, the geometric shapes detailed above can be extracted from a given point clouds data set. In order to improve the extraction process and reduce the number of points validating shape detection, we compute the approximated surface normal for each point and test the relevant shapes.

Given a point-clouds $P = \{p_1..p_N\}$ with associated normals $\{n_1..n_N\}$, the output of the RANSAC algorithm is a set of primitive shapes $\{\delta_1..\delta_N\}$ and a set of remaining points $R = P \setminus \{p_{\delta_1}..p_{\delta_N}\}$.

B. Predicted Scene – Kalman Filter

In this part, we present the global KF approach for our discrete dynamic system at the estimated state, *k*, based on the defined geometric shapes formulation defined in the previous sub-section.

Generally, the Kalman Filter can be described as a filter that consists of three major stages: Predict, Measure, and Update the state vector. The state vector contains different state parameters and provides an optimal solution for the whole dynamic system. We model our system as a linear one with discrete dynamic model, as described in (1):

$$x_k = F_{k,k-1} x_{k-1} \tag{1}$$

where x is the state vector, F is the transition matrix and *k* is the state.

The state parameters for all the geometric shapes are defined with shape center $\vec{s}$, and unit direction vector $\vec{d}$, of the geometric shape, from the current time step and viewpoint to the predicted one.

In each of the current states *k*, geometric shape center $\vec{s_k}$, is estimated based on the previous update of shape center location $\vec{s}_{k-1}$, and the previous updated unit direction vector $\vec{d}_{k-1}$, multiplied by small arbitrary scalar factor *c*, described in (2):

$$\vec{s}_k = \vec{s}_{k-1} + c\vec{d}_{k-1} \tag{2}$$

Direction vector $\vec{d}_k$ can be efficiently estimated by extracting the rotation matrix T, between the last two states *k, k-1*. In case of an inertial system fixed on the vehicle, a rotation matrix can be simply found from the last two states of the vehicle translations in (3):

$$\vec{d}_k = T\vec{d}_{k-1} \tag{3}$$

The 3D rotation matrix T tracks the continuous extracted plans and surfaces to the next viewpoint V_k, making it possible to predict a scene model where one or more of the geometric shapes are cut from current point clouds data in state *k-1*. The discrete dynamic system can be written as formulated in (4):

$$\begin{bmatrix} \vec{s}_{x_k} \\ \vec{s}_{y_k} \\ \vec{s}_{z_k} \\ \vec{d}_{x_k} \\ \vec{d}_{y_k} \\ \vec{d}_{z_k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & c & 0 & 0 \\ 0 & 1 & 0 & 0 & c & 0 \\ 0 & 0 & 1 & 0 & 0 & c \\ 0 & 0 & 0 & T_{11} & T_{12} & T_{13} \\ 0 & 0 & 0 & T_{21} & T_{22} & T_{23} \\ 0 & 0 & 0 & T_{31} & T_{32} & T_{33} \end{bmatrix} \begin{bmatrix} \vec{s}_{x_{k-1}} \\ \vec{s}_{y_{k-1}} \\ \vec{s}_{z_{k-1}} \\ \vec{d}_{x_{k-1}} \\ \vec{d}_{y_{k-1}} \\ \vec{d}_{z_{k-1}} \end{bmatrix} \tag{4}$$

where the state vector x is 6×1 vector, and the transition squared matrix is $F_{k,k-1}$. The dynamic system can be extended to additional state variables representing some of the geometric shape parameters such as radius, length etc. We define the dynamic system as the basic one for generic shapes that can be simply modeled with center and direction vector. Sphere radius and cylinder Z boundaries are defined in an additional data structure of the scene entities.

III. Fast and Approximated Visibility Analysis

In this section, we present an analytic analysis of the visibility boundaries of planes, cylinders and spheres for the predicted scene presented in the previous sub-section, which leads to an approximated visibility. For the plane surface, fast and efficient visibility analysis was already presented in [6]. In this part, we extend the previous visibility analysis concept [6] and include cylinders as continuous curves parameterization $C_{cln\,d}(x,y,z)$.

Cylinder parameterization can be described in (5):

$$C_{Cln\,d}(x,y,z)=\begin{pmatrix} r\sin(\theta) \\ r\cos(\theta) \\ c \end{pmatrix}_{r=const}, \quad \begin{matrix} 0\le\theta\le2\pi \\ c=c+1 \\ 0\le c\le h_{peds\_\max} \end{matrix} \tag{5}$$

We define the visibility problem in a 3D environment for more complex objects as:

$$C'(x,y)_{z_{const}}\times(C(x,y)_{z_{const}}-V(x_0,y_0,z_0))=0 \tag{6}$$

where 3D model parameterization is $C(x,y)_{z=const}$, and the viewpoint is given as $V(x_0,y_0,z_0)$. Extending the 3D cubic parameterization, we also consider the case of the cylinder. Integrating (5) to (6) yields:

$$\begin{pmatrix} r\cos\theta \\ -r\sin\theta \\ 0 \end{pmatrix}\times\begin{pmatrix} r\sin\theta-V_x \\ r\cos\theta-V_y \\ c-V_z \end{pmatrix}=0 \tag{7}$$

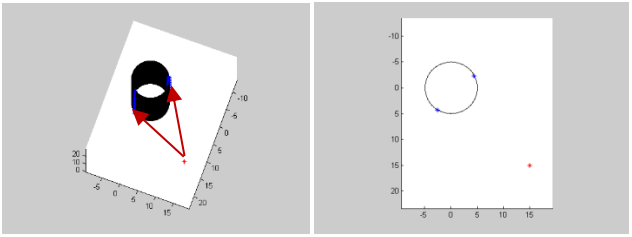
$$\theta=\arctan\left(\frac{-r-\frac{\left(-vy\,r+\sqrt{vx^4-vx^2\,r^2+vy^2\,vx^2}\right)vy}{vx^2+vy^2}}{vx},\ -\frac{-vy\,r+\sqrt{vx^4-vx^2\,r^2+vy^2\,vx^2}}{vx^2+vy^2}\right) \tag{8}$$

As can be noted, these equations are not related to Z axis, and the visibility boundary points are the same for each *x-y* cylinder profile, as seen in (7), (8).

The visibility statement leads to complex equation, which does not appear to be a simple computational task. This equation can be efficiently solved by finding where the equation changes its sign and crosses zero value; we used analytic solution to speed up computation time and to avoid numeric approximations. We generate two values of θ generating two silhouette points in a very short time computation. Based on an analytic solution to the cylinder case, a fast and exact analytic solution can be found for the visibility problem from a viewpoint.

We define the solution presented in (8) as x-y-z coordinates values for the cylinder case as Cylinder Boundary Points (CBP). CBP, defined in (9), are the set of visible silhouette points for a 3D cylinder, as presented in Figure 1:

$$CBP_{i=1..N_{PBP\_bound}=2}(x_0,y_0,z_0)=\begin{bmatrix} x_1,y_1,z_1 \\ x_{N_{PBP\_bound}},y_{N_{PBP\_bound}},z_{N_{PBP\_bound}} \end{bmatrix} \tag{9}$$



(a) (b)

Figure 1. Cylinder Boundary Points (CBP) using Analytic Solution marked as blue points, Viewpoint Marked in Red: (a) 3D View (Visible Boundaries Marked with Red Arrows); (b) Topside View.

In the same way, sphere parameterization can be described as formulated in (10):

$$C_{Sphere}(x,y,z)=\begin{pmatrix} r\sin\phi\cos\theta \\ r\sin\phi\sin\theta \\ r\cos\phi \end{pmatrix}_{r=const} \quad \begin{matrix} 0\le\phi<\pi \\ 0\le\theta<2\pi \end{matrix} \tag{10}$$

We define the visibility problem in a 3D environment for this object in (11):

$$C'(x,y,z)\ \times(C(x,y,z)\ -V(x_0,y_0,z_0))=0 \tag{11}$$

where the 3D model parameterization is $C(x,y,z)$, and the viewpoint is given as $V(x_0,y_0,z_0)$. Integrating (10) to (11) yields:

$$\theta = \arctan\left(\frac{r\sin(\phi)}{v\_y} - \frac{1}{v\_y\left(v\_y^2 + v\_x^2\right)}\left(v\_x\left(r\sin(\phi)\, v\_x - \sqrt{-v\_y^2 r^2 \sin(\phi)^2 + v\_y^4 + v\_x^2 v\_y^2}\right)\right), \frac{r\sin(\phi)\, v\_x - \sqrt{-v\_y^2 r^2 \sin(\phi)^2 + v\_y^4 + v\_x^2 v\_y^2}}{v\_y^2 + v\_x^2} \right) \quad (12)$$

Where r is defined from sphere parameter, and $V(x_0, y_0, z_0)$ are changes from visibility point along Z axis, as described in (12). The visibility boundary points for a sphere, together with the analytic solutions for planes and cylinders, allow us to compute fast and efficient visibility in a predicted scene from local point cloud data, which are updated in the next state.

This extended visibility analysis concept, integrated with a well-known predicted filter and extraction method, can be implemented in real time applications with point clouds data.

IV. DECENTRALIZED SWARMS TRAJECTORY PLANNING

In this part, we focus on decentralized swarm algorithms with visibility analysis in urban environment as cost function for our trajectory.

For our simulation, we used SwarmLab [8], drone swarm simulator that was implemented and adapted two representative algorithms belonging to the category of decentralized swarming. Decentralized approach can make the system easily scalable and robust to the failures of a single individual. SwarmLab includes algorithm developed by Olfati-Saber [9], who proposes a formal theoretical framework for the design and analysis of swarm algorithms based on potential fields and graph theory.

The second algorithm that was implemented is an adaptation of the recent Vasarhelyi's algorithm [10], defined by the following rules: repulsion to avoid inter-agent collisions, velocity alignment to steer the agents to an average direction, and self-propulsion to match a preferred speed value. In addition, the algorithm includes friction forces that reduce oscillations and ease the implementation on real robots.

In decentralized approaches, one agent's movement is only influenced by local information coming from its neighbors. Neighbors' selection can be operated according to different metrics.

In this paper, we adopted these algorithms with visibility analysis as part of swarm's trajectory by leading the swarm to the most visible areas in the scene by the swarm, as presented in the previous section.

Unlike the original SwarmLab simulation where obstacle avoidance is based on simulating the obstacles as virtual agents, we used the Velocity Obstacles [11] local obstacles avoidance method.

This obstacle avoidance method allows us to deal better with swarm behavior and can be more precise and gentler, avoiding obstacles in dense environments.

A. The Planner

As mentioned above, our planner is based on an iterative local planning method, where the swarm is moving to the most visible area. By using RANSAC algorithm, point clouds data are extracted at each time step into three possible objects: plane, cylinder and sphere. The scene is formulated as a dynamic system using KF analysis for objects' prediction. The objects are approximated for the next time step, and each safe attainable state that can be explored is set as candidate viewpoint. The cost for each agent is set as total visible surfaces, based on the analytic visibility boundary, where the optimal and safe node is explored for the next time step.

At each time step, the planner computes the next Attainable Velocities (AV). The safe nodes not colliding with objects such as cubes, cylinders and spheres, i.e., nodes outside VO, are explored. Where all nodes are inside VO, a unified analytic solution for time horizon is presented, generating an escape option for these radical cases without affecting visibility analysis. The planner computes the cost for these safe nodes based on predicted visibility and chooses the node with the optimal cost for the next time step. We repeat this procedure while generating the most visible trajectory.

B. Visibility Velocity Obstacles (VVO)

The visibility velocity obstacle represents the set of all velocities from a viewpoint, occluded with other objects in the environment. It essentially maps static and moving objects into the robot's velocity space considering visibility boundaries.

The VVO of an object with circular visibility boundary points such as the pedestrians' case, PBP, that is moving at a constant velocity v_b, is a cone in the velocity space at point A. In Figure 2, the position space and velocity space of A are overlaid to illustrate the relationship between the two spaces. The VVO is generated by first constructing the Relative Velocity Cone (RVC) from A to the boundaries of the object, i.e., PBP, then translating RVC by v_b.

Each point in VVO represents a velocity vector that originates at A. Any velocity of A that penetrates VVO is an occluded velocity that based on the current situation, would result in an occlusion between A and the pedestrian at some future time. Figure 2 shows two velocities of A: one that penetrates VVO, hence, an occluded velocity, and one that does not. All velocities of A that are outside of VVO are visible from the current robot's position as the obstacle denotes as B, stays on its current course.

The visibility velocity obstacle thus allows determining if a given velocity is occluded and suggesting possible changes to this velocity for better visibility. If PBP is known to move

along a curved trajectory or at varying speeds, it would be best represented by the nonlinear visibility velocity obstacle case discussed next.

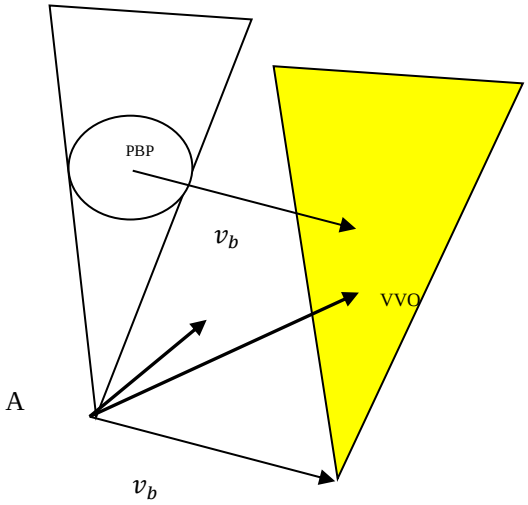


Figure 2. Visibility Velocity Obstacles

The VVO consists of all velocities of A at t_0 predicting visibility's boundaries related to obstacles at the environment at any time $t>t_0$. Selecting a single velocity, v_a, at time $t = t_0$ outside the VVO, guarantees visibility to this specific obstacle at time t. It is constructed as a union of its temporal elements, VVO(t), which is the set of all absolute velocities of A, v_a, that would allow visibility at a specific time t.

Referring to Figure 3, v_a that would result in occlusion with point p in B at time $t > t_0$, expressed in a frame centered at $A(t_0)$, is simply in (13):

$$v_a = \frac{VBP_i}{t-t_0} \quad (13)$$

where r is the vector to point p in the blocker's fixed frame, and visibility boundaries denoted as Visibility Boundary Points (VBP). The set VVO(t) of all absolute velocities of A that would result in occlusion with any point in B at time $t > t_0$ is thus in (14):

$$VVO(t) = \frac{VBP_i(t)}{t-t_0} \quad (14)$$

Clearly, VVO(t) is a scaled B for two-dimensional case with circular object, located at a distance from A that is inversely proportional to time t. The entire VVO is the union of its temporal subsets from t_0, the current time, to some set future time horizon t_h in (15):

$$VVO(t) = \bigcup_{t=t_0}^{t_h} \frac{VBP_i(t)}{t-t_0} \quad (15)$$

The presented VVO generates a warped cone in a case of 2D circular object. If VBP(t) is bounded over $t = (t_0, \infty)$, then the apex of this cone is at $A(t_0)$.We extend our analysis to 3D general case, where the objects can be cubes, cylinders and circles. The mathematical analysis with visibility boundaries is based on VBP presented in the previous part for different kind of objects such as buildings, cars and pedestrians.

We transform the visibility's boundaries into the velocity space, by moving the VBP to the velocity space, in the same analysis presented for 2D circle boundaries.

Following that, we present a 3D extension for VBP case, transformed to the velocity space.

Given two objects, VBP_1, VBP_2 will create a VVO representing VBP_2 (and vice-versa) such that VBP_1 wishes to choose a guaranteed collision-free velocity for the time interval τ, and visibility boundary in velocity space.

In case of cars, buildings and pedestrians where visibility boundaries can be expressed by geometric operations of 3D boxes, analyzed in the same concept and formulation presented so far, as can be seen in Figure 3.

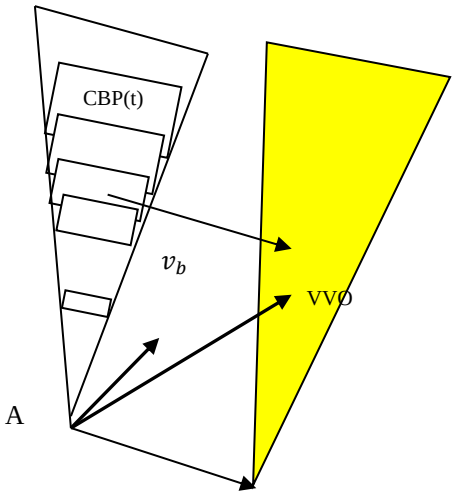


Figure 3. Visibility Velocity Obstacle for visibility boundaries consists of 3D boxes

C. Pursuer Planner Using VVO

Our planner, similar to previous work [11] is a local one, generating one step ahead every time step reaching toward the goal, which is a depth first A* search over a tree. We extend previous planners which take into account kinematic and dynamic constraints and present a local planner for UAV as case study with these constraints, which for the first time generates fast and exact visible trajectories based on VVO, tracking after a target by choosing the optimal next action based on velocity estimation. The fast and efficient visibility analysis of our method allows us to generate the most visible trajectory from a start state q_{start} to the goal state q_{goal} in 3D urban environments, which can be extended to real performances in the future. We assume knowledge of the 3D

urban environment model, and by using Visibility Velocity Obstacles (VVO) method to avoid occlusion, planner is based on exploring maximum visible node in the next time step and track a specific target.

1) ***Attainable Velocities***

Based on the dynamic and kinematic constraints, UAVs velocities at the next time step are limited. At each time step during the trajectory planning, we map the AV, the velocities set at the next time step $t+\tau$, which generate the optimal trajectory, as it is well-known from Dubins theory.

We denote the allowable controls as $u=(u_s,u_z,u_\phi)$ as U, where $V \in U$.

We denote the set of dynamic constraints bounding control's rate of change as $\dot{u}=(\dot{u}_s,\dot{u}_z,\dot{u}_\phi) \in U'$.

Considering the extremal controllers as part of the motion primitives of the trajectory cannot ensure time-optimal trajectory for Dubins airplane model but is still a suitable heuristic based on time-optimal trajectories of Dubin - car and point mass models.

We calculate the next time step's feasible velocities $\tilde{U}(t+\tau)$, between $(t,t+\tau)$ as shown in (16):

$$\tilde{U}(t+\tau) = U \cap \{u | u = u(t) \oplus \tau \cdot U'\} \quad (16)$$

Integrating $\tilde{U}(t+\tau)$ with UAV model yields the next eight possible nodes for the following combinations in (17):

$$\tilde{U}(t+\tau) = \begin{pmatrix} \tilde{U}_s(t+\tau) \\ \tilde{U}_z(t+\tau) \\ \tilde{U}_\phi(t+\tau) \end{pmatrix} = \begin{pmatrix} u_s^{\min}, u_s(t)+a_s\tau \\ -u_s^{\max}\tan\phi^{\max}, u_s(t)\tan u_\phi(t)+u_s^{\max}\tan a_\phi \\ u_z^{\max}, u_z(t)-a_z\tau \end{pmatrix} \quad (17)$$

At each time step, we explore the next eight AV at the next time step as part of our tree search, as explained in the next sub-section.

2) ***Tree Search***

Our planner uses a depth first A* search over a tree that expands over time to the goal. Each node $(q,\dot{q})$,where $q=(x,y,z,\theta)$, consists of the current UAVs position and velocity at the current time step. At each state, the planner computes the set of AV, $\tilde{U}(t+\tau)$, from the current UAV velocity, $U(t)$. We ensure the visibility of nodes by computing a set of Visibility Velocity Obstacles (VVO).

The search method is based on exploring nodes which are outside of VVO. The safe node with the lowest cost, which is the next most visible node, is explored in the next time step. This is repeated while generating the most visible trajectory, as discussed in the next sub-section.

Attainable velocities profile is similar to a trunked cake slice, due to the Dubins airplane model with one time step integration ahead. Simple models attainable velocities, such as point mass, create rectangular profile.

3) ***Cost Function***

Our swarm direction and movement is guided by minimum invisible parts from viewpoint *V* to the approximated 3D urban environment model in the next time step, $t+\Delta t$, set by KF after extracting objects from point clouds data using the RANSAC algorithm. The cost function next state is a combination of IRV and ISV, with different weights as functions of the required task.

The cost function presented in (18) is computed for each agent from its current state, considering the agent's future location at the next time step $(x_1(t+\Delta t), x_2(t+\Delta t))$ as viewpoint:

$$w\big(q(t+\tau)\big) = abs(v_a(q(t+\tau) - v_{tck}(q(t+\tau)) \quad (18)$$

where $\propto, \beta$ are coefficients affecting the trajectory's character, as shown in (14). The cost function $w(x(t+\Delta t)$ produces the total sum of invisible parts from the viewpoint to the 3D urban environment.

We divide point invisibility value into Invisible Surfaces Value (ISV) and Invisible Roofs Value (IRV). This classification allows us to plan delicate and accurate trajectories upon demand. We define ISV and IRS as the total sum of the invisible roofs and surfaces (respectively). Invisible Surfaces Value (ISV) of a viewpoint is defined as the total sum of the invisible surfaces of all the objects in a 3D environment, as described in (19):

$$ISV(x_0,y_0,z_0) = \sum_{i=1}^{N_{obj}} IS_{VP_i^{j=1..N_{bound}-1}}^{VP_i^{j=1..N_{bound}-1}} \quad (19)$$

In the same way, we define Invisible Roofs Value (IRV) as the total sum of all the invisible roofs' surfaces, as described in (20):

$$IRV(x_0,y_0,z_0) = \sum_{i=1}^{N_{obj}} IS_{VP_i^{j=N_{bound}}}^{VP_i^{j=N_{bound}}} \quad (20)$$

Extended analysis of the analytic solution for visibility analysis for known 3D urban environments can be found in [12].

V. SIMULATIONS

We implemented the presented algorithm and tested some urban environments on an 1.8GHz Intel Core CPU with

Matlab. We computed the visible trajectories using our planner, simulating cloud points using Matlab functions.

On the first part, we tested our visibility analysis integrated into decentralized drones swarm algorithms as described above. The workflow of a swarm simulation is summarized in Figure 4, were typical scenario of cylinder objects in our environment can be seen in Figure 5.

In the first case, we tested our algorithm with relatively large number of agents. As can be seen in Figure 4, thirty agents in the swarm moving forward in straight line, presenting swarm trajectory, distance between the agents during mission, speed and accelerations during movement. The swarm navigates based on modified Olfati-Saber's algorithm where obstacle avoidance implemented by Velocity Obstacles, where the agents are simulated by point mass model. Swarm cost function is based on visibility analysis computed each time step as mentioned in the previous section.

In the second case, we tested our algorithm with ten agents in the swarm, so each agent simulated with quadrotor dynamic model. As can be seen in Figure 6, ten agents in the swarm moving forward in straight line with Vasarhelyi's algorithm, but visibility analysis and dynamic constraints swift the swarm to the right side. presenting swarm trajectory, distance between the agents. Figure 5 also includes speed and accelerations during movement, performances analysis and total distance to the obstacles during mission.

Order metric captures the correlation of the agents movements and gives an indication about how ordered the flock. S*afety* metrics measure the risk of collisions among the swarm agents or between agents and obstacles.

Union metric counts the number of independent subgroups that originates during the simulation.

Connectivity metric is defined from the algebraic connectivity of the sensing graph that underlines the considered swarm configuration.

Detailed mathematical definitions of these performances' parameters can be found in [8].

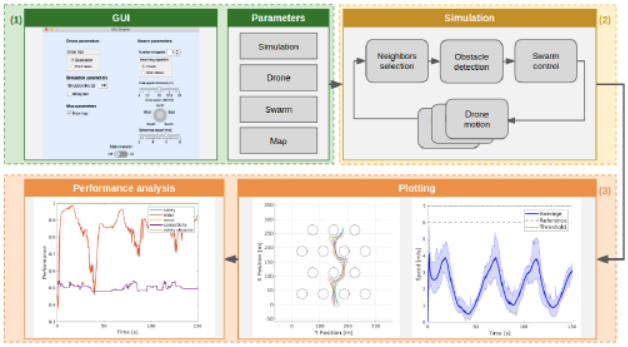


Figure 4. SwarmLab simulation workflow. From the top left, in clockwise order: (1) in the GUI, the user sets the parameters related to the simulation, drone typology, swarm algorithm and environment. (2) the main simulation loop computes control commands for the drones, based on the information of the map and neighboring drones; (3) both real-time and post-simulation (Source [8]).

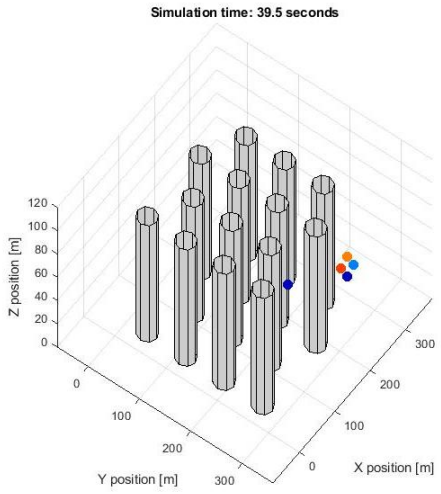
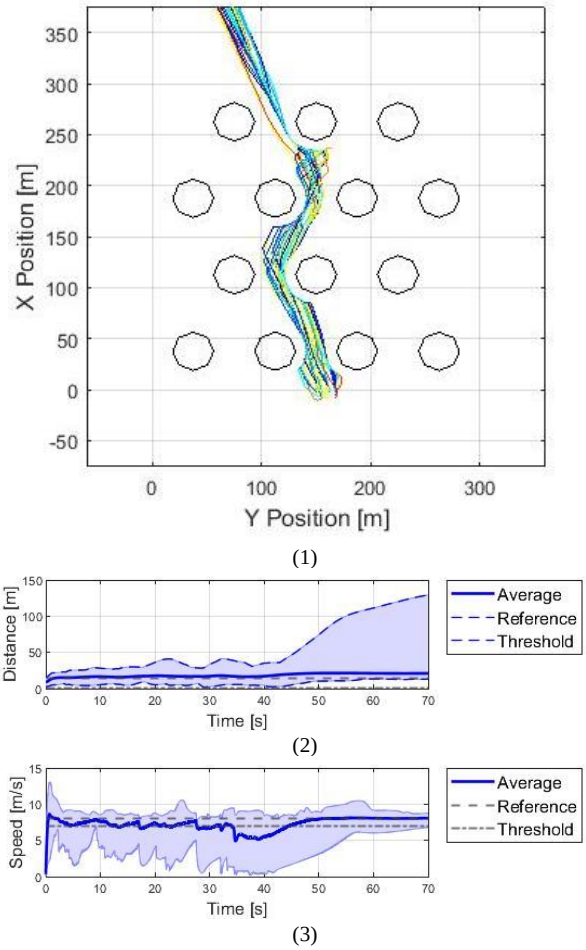
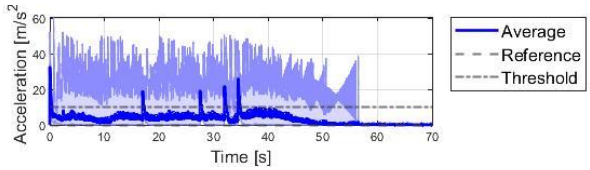


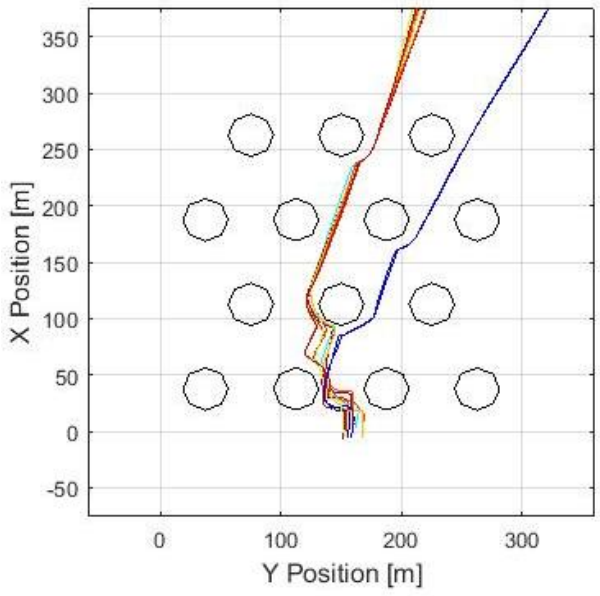
Figure 5. Typical Scenario of Environmmet Obstacles Simulation



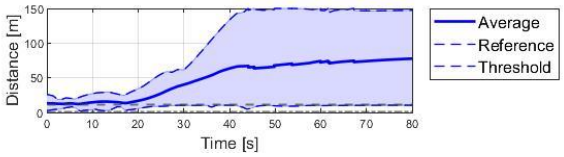


(4)

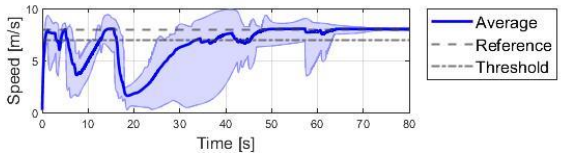
Figure 6. Thirty agents swarm moving forward in straight line using Olfati-Saber's algorithm with visibility analsysis; (1) presenting swarm trajectory; (2) distance between the agents during mission; (3) speed and (4) accelerations during movement.



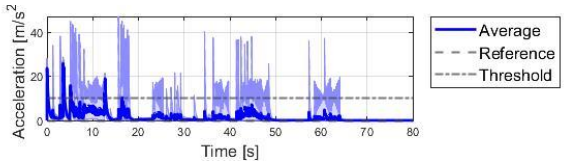
(1)



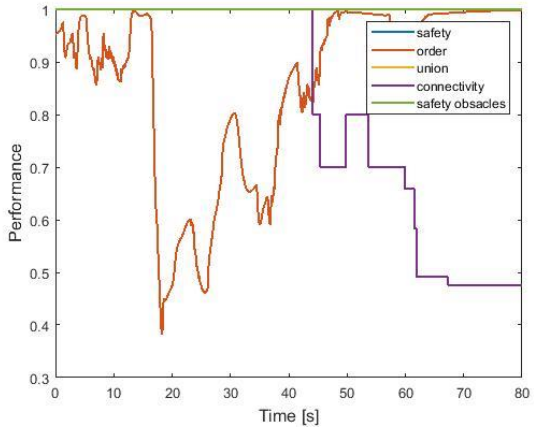
(2)



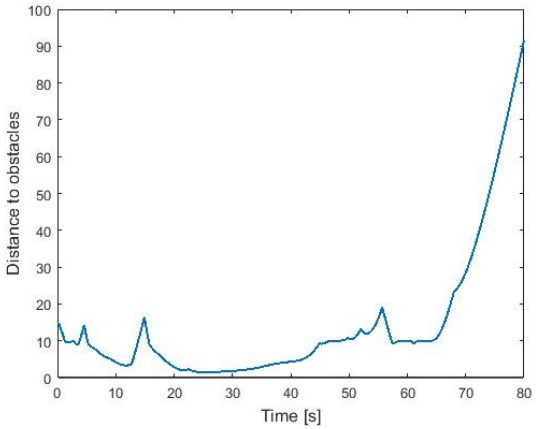
(3)



(4)



(5)



(6)

Figure 7. Ten agents swarm moving forward in straight line using Vasarhelyi's algorithm with visibility analsysis, with quadrotor synamic model for agent; (1) presenting swarm trajectory; (2) distance between the agents during mission; (3) speed and (4) accelerations during movement; (5) performances analysis; (6) total distance to the obstacles during mission.

VI. Conclusion and Future Work

In this research, we have presented an efficient swarm trajectory planning algorithm for visible trajectories in a 3D urban environment.

We extend our analytic visibility analysis method to cylinders and spheres, which allows us to efficiently set the visibility boundary of predicted objects in the next time step. Based on these fast computation capabilities, the on-line planner can approximate the most visible state as part of a decentralized swarm algorithms.

By using SwarmLab environment, we compare two decentralized algorithms from the state of the art for the navigation of aerial swarms, Olfati-Saber's and Vasarhelyi's.

Our planner includes dynamic and kinematic platform's limitation, generating visible trajectories based on our first step mentioned earlier.

We demonstrate our visibility and trajectory planning method in simulations, showing trajectory planning in 3D

urban environments for drone's swarm with decentralized algorithms with performance analysis such as order, safety, connectivity and union.

Further research will focus on advanced geometric shapes, which will allow precise urban environment modeling, facing real-time implementation with on-line data processing from sensors.

REFERENCES

[1] O. Gal and Y. Doytsher, "Decentralized Swarms Visibility Algorithms in 3D Urban Environments," GEOProcessing 2022, The Fourteenth International Conference on Advanced Geographic Information Systems, Applications, and Services, pp. 47-52.

[2] G. Vosselman, B. Gorte, G. Sithole and T. Rabbani. "Recognizing structure in laser scanner point clouds," The International Archives of the Photogrammetry Remote Sensing and Spatial Information Sciences (IAPRS), 2004, vol. 36, pp. 33–38.

[3] R. Schnabel, R. Wahl and R. Klein, "Efficient RANSAC for Point-Cloud Shape Detection," Computer Graphics Forum, 2007, vol. 26, no.2, pp. 214-226.

[4] R. Kalman. "A new approach to linear filtering and prediction problems," Transactions of the ASME-Journal of Basic Engineering, 1960, vol. 82, no. 1, pp:35–45.

[5] J. Lee, M. Kim and I. Kweon. "A Kalman filter based visual tracking algorithm for an object moving," In IEEE/RSJ Intelligent Robots and Systems, 1995, pp. 342–347.

[6] O. Gal and Y. Doytsher, "Fast Visibility Analysis in 3D Procedural Modeling Environments," in Proc. of the, 3rd International Conference on Computing for Geospatial Research and Applications, Washington DC, USA, 2012.

[7] H. Boulaassal, T. Landes, P. Grussenmeyer and F. Tarsha-Kurdi. "Automatic segmentation of building facades using terrestrial laser data," The International Archives of the Photogrammetry Remote Sensing and Spatial Information Sciences (IAPRS), 2007, vol. 36, no. 3.

[8] E. Soria, F. Schiano and D. Floreano, "SwarmLab: a Matlab Drone Swarm Simulator," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 8005-8011, doi: 10.1109/IROS45743.2020.9340854.

[9] R. Olfati-Saber, "Flocking for Multi-Agent Dynamic Systems: Algorithms and Theory," IEEE Transactions on Automatic Control, vol. 51, no. 3, pp. 401–420, 2006.

[10] G. V´as´arhelyi, C. Vir´agh, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek, "Optimized flocking of autonomous drones in confined environments," Science Robotics, vol. 3, no. 20, 2018.

[11] O. Gal, Z. Shiller and E. Rimon, "Efficient and safe on-line motion planning in dynamic environment," in Proceedings of the IEEE International Conference on Robotics and Automation, 2009, pp. 88–93.

[12] O. Gal and Y.Doytsher. "Patrolling Strategy Using Heterogeneous Multi Agents in Urban Environments Using Visibility Clustering," Journal of Unmanned System Technology, ISSN 2287-7320, 2016.